Experiment Number: 3

NAME: **Kartik Banshi Katkar** ROLLNO: 36

CLASS: TY IT A BATCH: 1

DATE OF PERFORMANCE: 03/08/2023

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Question:  
  
Write C-language code to read and Display BMP image. Perform following operations on the given image 1. Flip Image along horizontal and vertical direction 2. Crop the image 3. Convert gray scale image into Binary image 4. Rotate the given image by 45 degree angle.**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#pragma pack(2)

typedef struct {

unsigned short type;

unsigned int size;

unsigned short reserved1;

unsigned short reserved2;

unsigned int offset;

} BMPHeader;

typedef struct {

unsigned int size;

int width;

int height;

unsigned short planes;

unsigned short bitsPerPixel;

unsigned int compression;

unsigned int imageSize;

int xPixelsPerMeter;

int yPixelsPerMeter;

unsigned int colorsUsed;

unsigned int colorsImportant;

} BMPInfoHeader;

#pragma pack()

// Function declarations

void flipHorizontal(BMPHeader \*header, BMPInfoHeader \*infoHeader, unsigned char \*data);

void flipVertical(BMPHeader \*header, BMPInfoHeader \*infoHeader, unsigned char \*data);

void cropImage(BMPHeader \*header, BMPInfoHeader \*infoHeader, unsigned char \*data, int x, int y, int width, int height);

void convertToBinary(BMPHeader \*header, BMPInfoHeader \*infoHeader, unsigned char \*data);

void rotateImage(BMPHeader \*header, BMPInfoHeader \*infoHeader, unsigned char \*data);

int main() {

FILE \*bmpFile;

BMPHeader header;

BMPInfoHeader infoHeader;

unsigned char \*data = NULL;

int choice;

bmpFile = fopen("saulbmp2.bmp", "rb");

if (bmpFile == NULL) {

printf("Error: Unable to open the BMP file.\n");

return 1;

}

fread(&header, sizeof(BMPHeader), 1, bmpFile);

fread(&infoHeader, sizeof(BMPInfoHeader), 1, bmpFile);

data = (unsigned char \*)malloc(infoHeader.imageSize);

fseek(bmpFile, header.offset, SEEK\_SET);

fread(data, infoHeader.imageSize, 1, bmpFile);

fclose(bmpFile);

printf("Menu:\n");

printf("1. Flip Image along horizontal direction\n");

printf("2. Flip Image along vertical direction\n");

printf("3. Crop Image\n");

printf("4. Convert Gray Scale Image into Binary Image\n");

printf("5. Rotate the Image by 45 degrees\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

flipHorizontal(&header, &infoHeader, data);

break;

case 2:

flipVertical(&header, &infoHeader, data);

break;

case 3:

{

int x, y, width, height; // Declare variables within a separate block

printf("Enter the x-coordinate of the top-left corner: ");

scanf("%d", &x);

printf("Enter the y-coordinate of the top-left corner: ");

scanf("%d", &y);

printf("Enter the width of the cropped image: ");

scanf("%d", &width);

printf("Enter the height of the cropped image: ");

scanf("%d", &height);

cropImage(&header, &infoHeader, data, x, y, width, height);

}

break;

case 4:

convertToBinary(&header, &infoHeader, data);

break;

case 5:

rotateImage(&header, &infoHeader, data);

break;

default:

printf("Invalid choice.\n");

free(data);

return 1;

}

bmpFile = fopen("output.bmp", "wb");

if (bmpFile == NULL) {

printf("Error: Unable to create output BMP file.\n");

free(data);

return 1;

}

fwrite(&header, sizeof(BMPHeader), 1, bmpFile);

fwrite(&infoHeader, sizeof(BMPInfoHeader), 1, bmpFile);

fseek(bmpFile, header.offset, SEEK\_SET);

fwrite(data, infoHeader.imageSize, 1, bmpFile);

fclose(bmpFile);

free(data);

printf("Operation completed successfully.\n");

return 0;

}

void flipHorizontal(BMPHeader \*header, BMPInfoHeader \*infoHeader, unsigned char \*data) {

int bytesPerPixel = infoHeader->bitsPerPixel / 8;

int rowSize = infoHeader->width \* bytesPerPixel;

unsigned char \*tempRow = (unsigned char \*)malloc(rowSize);

for (int y = 0; y < infoHeader->height; y++) {

int offset = y \* rowSize;

for (int x = 0; x < infoHeader->width / 2; x++) {

int leftPixelOffset = offset + x \* bytesPerPixel;

int rightPixelOffset = offset + (infoHeader->width - x - 1) \* bytesPerPixel;

// Swap pixels (left and right)

for (int i = 0; i < bytesPerPixel; i++) {

unsigned char temp = data[leftPixelOffset + i];

data[leftPixelOffset + i] = data[rightPixelOffset + i];

data[rightPixelOffset + i] = temp;

}

}

}

free(tempRow);

}

void flipVertical(BMPHeader \*header, BMPInfoHeader \*infoHeader, unsigned char \*data) {

int bytesPerPixel = infoHeader->bitsPerPixel / 8;

int rowSize = infoHeader->width \* bytesPerPixel;

unsigned char \*tempRow = (unsigned char \*)malloc(rowSize);

for (int y = 0; y < infoHeader->height / 2; y++) {

int topOffset = y \* rowSize;

int bottomOffset = (infoHeader->height - y - 1) \* rowSize;

// Swap rows (top and bottom)

memcpy(tempRow, data + topOffset, rowSize);

memcpy(data + topOffset, data + bottomOffset, rowSize);

memcpy(data + bottomOffset, tempRow, rowSize);

}

free(tempRow);

}

void cropImage(BMPHeader \*header, BMPInfoHeader \*infoHeader, unsigned char \*data, int x, int y, int width, int height) {

int bytesPerPixel = infoHeader->bitsPerPixel / 8;

int rowSize = infoHeader->width \* bytesPerPixel;

int croppedWidth = width;

int croppedHeight = height;

int xOffset = x \* bytesPerPixel;

if (x + width > infoHeader->width) {

croppedWidth = infoHeader->width - x;

}

if (y + height > infoHeader->height) {

croppedHeight = infoHeader->height - y;

}

unsigned char \*croppedData = (unsigned char \*)malloc(croppedWidth \* croppedHeight \* bytesPerPixel);

for (int row = 0; row < croppedHeight; row++) {

int srcOffset = (y + row) \* rowSize + xOffset;

int destOffset = row \* croppedWidth \* bytesPerPixel;

memcpy(croppedData + destOffset, data + srcOffset, croppedWidth \* bytesPerPixel);

}

// Update the BMP header and info header to match the cropped image dimensions

int imageSize = croppedWidth \* croppedHeight \* bytesPerPixel;

infoHeader->width = croppedWidth;

infoHeader->height = croppedHeight;

infoHeader->imageSize = imageSize;

header->size = header->offset + imageSize;

// Free the original data and update the data pointer to the cropped data

free(data);

data = croppedData;

}

void convertToBinary(BMPHeader \*header, BMPInfoHeader \*infoHeader, unsigned char \*data) {

int bytesPerPixel = infoHeader->bitsPerPixel / 8;

int rowSize = infoHeader->width \* bytesPerPixel;

for (int y = 0; y < infoHeader->height; y++) {

int offset = y \* rowSize;

for (int x = 0; x < infoHeader->width; x++) {

int pixelOffset = offset + x \* bytesPerPixel;

unsigned char grayscaleValue = data[pixelOffset];

unsigned char binaryValue = (grayscaleValue < 128) ? 0 : 255;

for (int i = 0; i < bytesPerPixel; i++) {

data[pixelOffset + i] = binaryValue;

}

}

}

}

void rotateImage(BMPHeader \*header, BMPInfoHeader \*infoHeader, unsigned char \*data) {

int bytesPerPixel = infoHeader->bitsPerPixel / 8;

int newRowSize = (int)(infoHeader->width \* 0.70710678); // cos(45 degrees) = sin(45 degrees) = 0.70710678

int newColSize = newRowSize;

// Create a new buffer for the rotated image

unsigned char \*rotatedData = (unsigned char \*)malloc(newRowSize \* newColSize \* bytesPerPixel);

int centerX = infoHeader->width / 2;

int centerY = infoHeader->height / 2;

for (int y = 0; y < newColSize; y++) {

for (int x = 0; x < newRowSize; x++) {

int rotatedX = x - centerX;

int rotatedY = y - centerY;

// Apply rotation transformation

int srcX = (int)(rotatedX \* 0.70710678 + rotatedY \* 0.70710678) + centerX;

int srcY = (int)(rotatedY \* 0.70710678 - rotatedX \* 0.70710678) + centerY;

// Perform bilinear interpolation to get the pixel value

int x0 = floor(srcX);

int x1 = ceil(srcX);

int y0 = floor(srcY);

int y1 = ceil(srcY);

int srcOffset00 = (y0 \* infoHeader->width + x0) \* bytesPerPixel;

int srcOffset01 = (y0 \* infoHeader->width + x1) \* bytesPerPixel;

int srcOffset10 = (y1 \* infoHeader->width + x0) \* bytesPerPixel;

int srcOffset11 = (y1 \* infoHeader->width + x1) \* bytesPerPixel;

int destOffset = (y \* newRowSize + x) \* bytesPerPixel;

// Perform bilinear interpolation for each channel

for (int i = 0; i < bytesPerPixel; i++) {

double dx = srcX - x0;

double dy = srcY - y0;

double pixelValue = (1 - dx) \* (1 - dy) \* data[srcOffset00 + i] +

dx \* (1 - dy) \* data[srcOffset01 + i] +

(1 - dx) \* dy \* data[srcOffset10 + i] +

dx \* dy \* data[srcOffset11 + i];

rotatedData[destOffset + i] = (unsigned char)pixelValue;

}

}

}

// Update the BMP header and info header to match the rotated image dimensions

int imageSize = newRowSize \* newColSize \* bytesPerPixel;

infoHeader->width = newRowSize;

infoHeader->height = newColSize;

infoHeader->imageSize = imageSize;

header->size = header->offset + imageSize;

// Free the original data and update the data pointer to the rotated data

free(data);

data = rotatedData;

}

**Input:   
  
 **

**Output:**